AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently Amended) A method for executing a commit instruction to facilitate transactional execution on a processor, comprising:

executing a block of instructions transactionally, wherein executing the block of instructions transactionally involves placing load-marks on cache lines from which data is loaded and placing store-marks on cache lines to which data is stored;

encountering the commit instruction during execution of a program, wherein the commit instruction marks the end of a block of instructions to be executed transactionally; and

upon encountering the commit instruction, successfully completing transactional execution of the block of instructions preceding the commit instruction, wherein successfully completing the transactional execution involves atomically committing changes made during the transactional execution by:

treating store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines;

committing store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves removing the store-mark from, and thereby unlocking, a corresponding store-marked cache line;

clearing load-marks from cache lines; and committing register file changes made during transactional execution;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

2. (Previously Presented) The method of claim 1, wherein successfully completing the transactional execution involves:

resuming normal non-transactional execution.

3. (Cancelled)

4. (Original) The method of claim 1, wherein if an interfering data access from another process is encountered during the transactional execution and prior to encountering the commit instruction, the method further comprises:

discarding changes made during the transactional execution; and

attempting to re-execute the block of instructions.

5. (Previously Presented) The method of claim 1, wherein for a variation of the commit instruction, successfully completing the transactional execution involves:

commencing transactional execution of the block of instructions following the commit instruction.

6. (Original) The method of claim 1, wherein potentially interfering data accesses from other processes are allowed to proceed during the transactional execution of the block of instructions.

7. (Original) The method of claim 1, wherein the block of instructions to be executed transactionally comprises a critical section.

8. (Original) The method of claim 1, wherein the commit instruction is a native machine code instruction of the processor.

9. (Original) The method of claim 1, wherein the commit instruction is defined in a platform-independent programming language.

10. (Currently Amended) A computer system that supports a commit instruction to facilitate transactional execution, wherein the commit instruction marks the end of a block of instructions to be executed transactionally, comprising:

a processor; and

an execution mechanism within the processor<u>, wherein the execution mechanism is configured to place load-marks on cache lines from which data is loaded and place store-marks on cache lines to which data is stored during transactional execution</u>;

wherein upon encountering the commit instruction, the execution mechanism is configured to successfully complete transactional execution of the block of instructions preceding the commit instruction, wherein successfully completing the transactional execution involves atomically committing changes made during the transactional execution by:

treating store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines;

committing store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves removing the store-mark from, and thereby unlocking, a corresponding store-marked cache line;

clearing load-marks from cache lines; and

committing register file changes made during transactional execution;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

11. (Previously Presented) The computer system of claim 10, wherein while successfully completing transactional execution, the execution mechanism is configured to:

resume normal non-transactional execution.

12. (Cancelled)

13. (Original) The computer system of claim 10, wherein if an interfering data access from another process is encountered during the transactional execution and prior to encountering the commit instruction, the execution mechanism is configured to:

discard changes made during the transactional execution; and to attempt to re-execute the block of instructions.

14. (Previously Presented) The computer system of claim 10, wherein if a variation of the commit instruction is encountered, the execution mechanism is configured to:

commence transactional execution of the block of instructions following the commit instruction.

15. (Original) The computer system of claim 10, wherein the computer system is configured to allow potentially interfering data accesses from other processes to proceed during the transactional execution of the block of instructions.

16. (Original) The computer system of claim 10, wherein the block of instructions to be executed transactionally comprises a critical section.

17. (Original) The computer system of claim 10, wherein the commit instruction is a native machine code instruction of the processor.

18. (Original) The computer system of claim 10, wherein the commit instruction is defined in a platform-independent programming language.

19. (Currently Amended) A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for executing a commit instruction to facilitate transactional execution, comprising:

<u>executing a block of instructions transactionally, wherein executing the block of instructions transactionally involves placing load-marks on cache lines</u>

<u>from which data is loaded and placing store-marks on cache lines to which data is stored;</u>

encountering the commit instruction during execution of a program, wherein the commit instruction marks the end of a block of instructions to be executed transactionally; and

upon encountering the commit instruction, successfully completing transactional execution of the block of instructions preceding the commit instruction, wherein successfully completing the transactional execution involves atomically committing changes made during the transactional execution by:

treating store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines;

committing store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves removing the store-mark from, and thereby unlocking, a corresponding store-marked cache line;

clearing load-marks from cache lines; and

committing register file changes made during transactional execution;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

20. (Previously Presented) The computer-readable storage medium of claim 19, wherein successfully completing transactional execution involves:

resuming normal non-transactional execution.